Using the Library

Mobile SDK: iOS

Creating a Project

The fastest way to get started is to check out the PaymentGatewayEncryptionExample and PaymentGatewaySwipeExample projects that can be downloaded from the Payment Gateway's Integration section. If you prefer to create your own project instead, use these steps (current as of Xcode 6.0):

1. Download the Mobile SDK .zip file from the Integration Portal by using the "Downloads" link under the Mobile SDK section. This file contains both the iOS and Android libraries.
2. Create a new Xcode Project.
3. Add PGMobileSDK.a and PGMobileSDK directory containing the headers to your project. These are found in the .zip file under Apple iOS -> Payment Gateway SDK.
4. Under the project's Build Phases settings, add these libraries to the Link Binary With Libraries section:
    - AudioToolbox.framework
    - AVFoundation.framework
    - ExternalAccessory.framework
    - MediaPlayer.framework
    - Security.framework
    - CoreBluetooth.framework
    - libstdc++.6.0.9.dylib
    - PGMobileSDK.a
5. (Optional - see note below) - In your Info.plist, add a row for "Supported external accessory protocols", and add "com.gatewayprocessingservices.iprocess" as Item 0. This enables connection to the iDynamo swipe reader.

Note: You may wish to skip step 5 if you do not need to support the iDynamo. Apple requires manufacturers of accessories that use the dock connector to add your app to their product plan before approving your app for the app store. You will need to contact MagTek in order to have your app added to their product plan. Contact MagTek for more details.

Viewing documentation in Xcode

Adding the doc set to Xcode allows the most up-to-date, relevant documentation to appear in the IDE as you type. To enable access to the SDK documentation from inside Xcode:

1. Under the Xcode menu, click Preferences
2. Navigate to the Downloads page
3. On the Documentation tab, click Add.
4. On the "Enter a doc set feed URL" window that pops up, enter: https://secure.safewebservices.com/merchants/resources/integration/docset/iOSSDK.atom

5. Click Add
6. Click the newly-added install button

Important Info About the App Store

The Apple App Store's current policy is to require mobile apps to purchase digital goods (e.g. downloadable content, etc.) through the App Store. For that reason, this SDK is intended only for use in apps selling real-world goods and services. Please direct questions about Apple's App Store policies to Apple. Their policies are subject to change at their discretion.

End-to-End Encryption

Mobile SDK: iOS

Acquiring a Public Key

a. After logging into the Payment Gateway, navigate to Settings->Security Keys->View Mobile SDK Key. You can click on the Objective-C example link to get a version that can easily be copied and pasted into your project.
b. Use the Query API. In order to get the public key, you will need to use 'report_type=sdk_key'. The key will be returned in the <sdk_key> tag.

Encrypting a Card

```
#import "PGEncrypt.h"
#import "PGCards.h"

PGEncrypt encryption = [[PGEncrypt alloc] init];
[encryption setKey:
    @"***999|MIIEEjCCA3ugAwIBAgIBADANBgkqhkiG9w0BAQQFADCBvTELMAkGA1UEBh"
    "MCVVMxETAPBgNVBAgTCElsbGlub2lzMRMwEQYDVQQHEwpTY2hhdW1idXJnMRgwFg"
                    [Several lines omitted]
    "cNAQEEBQADgYEAKY8xYc91ESNeXZYTVxEsFA9twZDpRjSKShDCcbutgPlC0XcHUt"
    "a2MfFPsdgQoq0I8y1nEn1qJiOuEG1t9Uwux4GAvAPzsWSsKyKQkZhqxrxkJUB39K"
    "Pg57pPytfJnlQTgYiSrycCEVHdDvhk92X7K2cab3aVV1+j0rKlR/Sy6b4=***"];


PGCard *cardData = [[PGKeyedCard alloc]
initWithCardNumber:cardNumberField.text

expirationDate:expirationField.text
                                                        cvv:cvvField.text];

NSString *encryptedCardData = [encryption encrypt:cardData includeCVV:NO];
```

encryptedCardData will contain a string that can be passed to the Payment Gateway in place of credit card information. The parameter name to use when passing this value through DirectPost is "encrypted_payment". For example, a simple DirectPost API string would look something like this:

(This example assumes your Merchant server is running a PHP script that has received the encrypted card data through a POST parameter called 'cardData'.)

```
//Business logic, validation, etc.  When ready to process the payment...
$cardData = $_POST['cardData'];
$postString =
"username=demo&password=1234&type=sale&amount=1.00&encrypted_payment=$cardDat
a";

//Post to Gateway
```

For more information on how to communicate with the Payment Gateway, see the API documentation.

Swipe Devices

Mobile SDK: iOS

Creating the Controller

In the class that intends to handle swipe events, create a PGSwipeController object in your init method. Initialize the object with one of these lines to support specific readers:

```
swipeController = [[PGSwipeController alloc] initWithDelegate:self
audioReader:AudioJackReaderIpsEnterprise];
swipeController = [[PGSwipeController alloc] initWithDelegate:self
audioReader:AudioJackReaderIps];
swipeController = [[PGSwipeController alloc] initWithDelegate:self
audioReader:AudioJackReaderUnimag];
```

Only a single model of audio jack-connected reader can be enabled at a time. The audioReader parameter allows you to choose which type you want to allow. See the PGSwipeController's initWithDelegate:audioReader: documentation for more details.

Your class will have to implement the PGSwipeDelegate protocol. If you are only interested in knowing when a card is swiped, you can safely leave every other event handler empty, as shown here (or add your own code to, for example, display an image indicating that the swipe reader is ready for a swipe). In this example, when the swipe is received, the card data is saved in a property (swipedCard) for eventual transmission to the Gateway (not shown), and two UITextField properties (cardNumberField and expirationField) are set to show the masked card number and expiration date.

If a bad swipe occurs, didSwipeCard:device: may still be called, but "card" will be nil. An error message is displayed in this example. Note: Not all card reader models give feedback when a bad swipe is received.

```
-(void)deviceConnected:(PGSwipeDevice *)sender
{
```

```
}

-(void)deviceDisconnected:(PGSwipeDevice *)sender
{
}

-(void)deviceActivationFinished:(PGSwipeDevice *)sender
result:(SwipeActivationResult)result
{
}

-(void)deviceDeactivated:(PGSwipeDevice *)sender
{
}

-(void)deviceBecameReadyForSwipe:(PGSwipeDevice *)sender
{
}

-(void)deviceBecameUnreadyForSwipe:(PGSwipeDevice *)sender
reason:(SwipeReasonUnreadyForSwipe)reason;
{
}

-(void)didSwipeCard:(PGSwipedCard *)card device:(PGSwipeDevice *)sender
{
    if (card != nil) {

        swipedCard = [card retain];

        cardNumberField.text = card.maskedCardNumber;
        expirationField.text = card.expirationDate;

    } else {

        //A nil card means that there was a swipe but it was unsuccessful.
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Swipe
Error"
                                                        message:@"The reader
was not able to read the card. Please Try Again."
                                                       delegate:nil
                                              cancelButtonTitle:@"OK"
                                              otherButtonTitles:nil];

        [alert show];
        [alert release];
    }

}
```

Classes Overview

Mobile SDK: iOS

PGSwipeController

The PGSwipeController contains a set of swipe reader classes that control individual swipe readers. This is the main Mobile Swipe SDK class required for using swipe devices, intended to be instantiated near the app's startup. The delegate you set on the PGSwipeController is the object that will receive all of the SDK's swipe events.

Through this class, you can access the controller classes for individual swipe device types (PGSwipeIpsEnterprise * ipsEnterpriseReader, PGSwipeIps *ipsReader, PGSwipeIDynamo *iDynamoReader, and PGSwipeUniMag *uniMagReader).

You should be sure to call initWithDelegate rather than the parameterless init because during initialization a check is made to see if any devices are already connected and sends a deviceConnected event if they are. If the parameterless init is used, the initial connection message will be missed.

- *-(id)initWithDelegate:(id<PGSwipeDelegate>)delegate audioReader:(AudioJackReaderType)readerType*

  Initializes the PGSwipeController and the individual swipe reader classes. Init checks if any devices are connected and sends a deviceConnected event if they are, so initWithDelegate: should always be used rather than init to ensure that a connection event is received if the device is already connected.

  The audioReader: parameter selects which type of audio jack-connected card reader to enable. Only one type of audio jack-connected reader can be used at a time to prevent more than one device library from attempting to access the audio system at the same time. AudioJackReaderIpsEnterprise enables the IPS Enterprise Encrypted Card Reader library, AudioJackReaderIps enables the IPS Encrypted Card Reader library, and AudioJackReaderUnimag enables the Shuttle library. You may also select AudioJackReaderNone to disable both libraries, or AudioJackReaderAutodetectOnConnect to allow the SDK to attempt to determine the type on connection. Autodetection has several drawbacks. See PGSwipeController beginAutodetectAudioJackCardReader for more information.

- *-(void)setAudioJackReaderType:(AudioJackReaderType)audioJackReaderType messageOptions:(PGSwipeUniMagMessageOptions *)messageOptions*

  Sets the enabled audioJackReaderType. This can be used to enable support for either the IPS Enterprise Encrypted Card Reader, IPS Encrypted Card Reader, or the UniMag (Shuttle) reader. Since the underlying libraries may not always unload cleanly, you should avoid calling this repeatedly to change the supported device type. Doing so could cause the reader to malfunction or be damaged. Setting this to AudioJackReaderAutodetectOnConnect will disable any currently selected audioJackReaderType and autodetect upon device connection.

messageOptions will be used only when audioJackReaderType is AudioJackReaderUnimag to replace the default message options. For any other AudioJackReaderType, or to use the default message options for AudioJackReaderUnimag, this should be nil.

- *-(void)beginAutodetectAudioJackCardReader;*

  Asynchronously attempts to detect the card reader type currently attached to the audio jack.

  A communication test is first attempted for an IPS Enterprise reader, then IPS reader and finally, an attempt is made to power up a UniMag (Shuttle) card reader. If either test succeeds, the audioJackReaderType is set to the correct value, and the device will be made ready for use. The result of the autodetect is reported to the delegate through deviceAutodetectComplete:.

  Note: All tests produce very loud tones through the audio jack. If speakers or headphones are attached, the tones would be unpleasant to the user. It is recommended that the user be warned and allowed to remove headphones before calling this function. This library suppresses user notifications from the UniMag reader during autodetect.

  Because the device is powered up in order to test it, you will not receive connection / activation / ready for swipe events during detection. When your delegate receives its deviceAutodetectComplete message, check the isConnected, isActive, and isReadyForSwipe properties for its current state and to complete any initialization.

  In order to detect the devices, all of the underlying card reader libraries must be loaded. Under some circumstances, these libraries may not unload cleanly, resulting in unreliable use of the card reader. Autodetect is also a very slow process. For these reasons, you should not rely on autodetection for each use of the app.

  Because communication through the audio jack is not always perfect, autodetect does not always return a correct result. The most common failure type is returning CardReaderAutodetectResultFail even though a supported device is connected.

  If it is known in advance which card reader type will be used, that type should be specified when initializing the PGSwipeController. If multiple devices must be supported, it is strongly recommended that the result of the autodetect be saved (e.g. in NSUserDefaults) and re-used on app startup.

PGSwipeDevice

The PGSwipeDevice class represents the functionality that is common to the swipe reader devices.

A PGSwipeDevice object is passed along with every event generated by the swipe devices to allow you to identify the device type and access device-specific features by casting to the specific swipe type.

- *bool isConnected*

  True when the reader is physically attached to the device.

- *bool isActivated*

  True when the reader is powered up / initialized.

- *bool isReadyForSwipe*

  True when the reader is able to accept card swipes from the user.

- *id<PGSwipeDelegate>delegate*

  Sets the delegate that will receive the device's events. You should not set the delegate directly. Setting the delegate on the PGSwipeController sets the delegate for each of its members.

PGSwipeDelegate

The PGSwipeDelegate protocol must be implemented by the class that intends to receive swipe reader events. The following event handlers will need to be implemented.

- *-(void)didSwipeCard:(PGSwipedCard *)card device:(PGSwipeDevice *)sender*

  This event is sent whenever the user swipes a card. Normally "card" will be either a PGEncryptedSwipedCard or PGMagnesafeSwipedCard (depending on the swipe reader) with track data, masked card number, expiration date, and cardholder name. If the swiped card cannot be read, "card" will be nil.

- *-(void)deviceBecameReadyForSwipe:(PGSwipeDevice *)sender*

  This event is sent when isReadyForSwipe becomes true. Between this event and the receipt of deviceBecameUnreadyForSwipe, any swipe should produce a didSwipeCard event.

- *-(void)deviceBecameUnreadyForSwipe:(PGSwipeDevice *)sender reason:(SwipeReasonUnreadyForSwipe)reason*

  This event is sent when isReadyForSwipe becomes false. There are many reasons the reader could become unready to receive swipe events, e.g. the swipe request times out, the device is disconnected, etc. Check the value of "reason" to determine the cause.

  On Shuttle readers, if you have set the device to alwaysAcceptSwipe, the reason may be set to SwipeReasonUnreadyForSwipeRefreshing. In that case, there is no need to request a new swipe. The "unready" state is momentary while the device automatically renews after a timeout, swipe, or other event.

- *-(void)deviceConnected:(PGSwipeDevice *)sender;*

  Occurs when the reader is physically connected to the device. With audio-port connected devices, this event can be sent when the user attaches headphones.

- *-(void)deviceDisconnected:(PGSwipeDevice *)sender;*

  Occurs when the reader is physically disconnected from the device. With audio-port connected devices, this event can be sent when the user detaches headphones.

- *-(void)deviceActivationFinished:(PGSwipeDevice *)sender result:(SwipeActivationResult)result;*

  Occurs when the device has finished an attempt to power up/initialize. This may occur at the same time as deviceConnected or later, depending on the device and settings. See the individual device documentation for specifics.

  Receiving this event does not mean the initialization succeeded. Be sure to check the value of "result" to verify that it is SwipeActivationResultSuccess.

- *-(void)deviceDeactivated:(PGSwipeDevice *)sender;*

  Occurs when the device has powered down. This may occur when the device is disconnected or, for certain swipe readers, when you make a call to power down the device.

- *-(void)deviceAutodetectStarted;*

  Occurs when an attempt to detect the type of an audio jack-connected card reader has started. This can be triggered by a manual call to PGSwipeController beginAutodetectAudioJackCardReader or automatically when the PGSwipeController is in AudioReaderAutodetectOnConnect mode and an object is attached to the audio jack by the user.

- *-(void)deviceAutodetectComplete:(CardReaderAutodetectResult)result;*

  Occurs when an attempt to detect the type of an audio jack-connected card reader has finished. The result may be CardReaderAutodetectResultIpsEnterprise,CardReaderAutodetectResultIps, CardReaderAutodetectResultUniMag, or CardReaderAutodetectResultFail if the type could not be determined.

  When this message is received, the PGSwipeController's audioJackReaderType will have been set to the appropriate value and the card reader will be activated. Since the device is powered up while autodetecting, no events for connection, activation, or readyForSwipe will be received. Check the isConnected, isActivated, and isReadyForSwipe properties to determine the device's state.

PGSwipeIpsEnterprise

This class is the interface to the IPS Enterprise Encrypted Card Reader. It is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create a PGSwipeIpsEnterprise instance to interact with the IPS Enterprise device.

- *id delegate;*

  Gets or sets the delegate that will receive events. You should not set this directly. When the PGSwipeController's delegate is set, it will pass it through to this delegate.

- *-(void)shutdown;*

  Closes the card reader's connections and disables event handling to allow it to be deallocated. You should not call this directly. It is called by the PGSwipeController when necessary.

- *-(void)beginTestCommunication(id)delegate;*

  Asynchronously sends a communication test message to the card reader device and waits for a response. This can be used to detect whether the connected device is an IPS Enterprise Encrypted Card Reader. A message is sent via the audio jack and if no response is received from the device within 5 seconds, the attached object is assumed not to be an IPS Enterprise reader. Note that calling this will produce a short, loud tone through the audio jack if headphones are attached.

  The result is returned to the delegate by calling ipsEnterpriseCommunicationTestCompleteWithResult:(BOOL)succes with success set to YES or NO, depending on if a response was sent by the device.

PGSwipeIps

This class is the interface to the IPS Encrypted Card Reader. It is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create a PGSwipeIps instance to interact with the IPS device.

- *id delegate;*

  Gets or sets the delegate that will receive events. You should not set this directly. When the PGSwipeController's delegate is set, it will pass it through to this delegate.

- *-(void)shutdown;*

Closes the card reader's connections and disables event handling to allow it to be deallocated. You should not call this directly. It is called by the PGSwipeController when necessary.

- *-(void)beginTestCommunication(id)communicationTestDelegate;*

  Asynchronously sends a communication test message to the card reader device and waits for a response. This can be used to detect whether the connected device is an IPS Encrypted Card Reader. A message is sent via the audio jack and if no response is received from the device within 5 seconds, the attached object is assumed not to be an IPS reader. Note that calling this will produce a short, loud tone through the audio jack if headphones are attached.

  The result is returned to the communcationTestDelegate by calling ipsCommunicationTestCompleteWithResult:(BOOL)succes with success set to YES or NO, depending on if a response was sent by the device.

PGSwipeIDynamo

This class is the interface to the iDynamo reader. It is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create a PGSwipeIDynamo instance to interact with the iDynamo device.

The iDynamo has no configurable options. When the device is attached, it is active and ready for swipe. The only property for the PGSwipeIDynamo class is a delegate to receive events, which should not be set directly. When the delegate is set for the PGSwipeController, the same delegate is passed to the PGSwipeIDynamo instance it contains.

PGSwipeUniMag

This class is the interface to the IDTECH Shuttle reader. It is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create a PGSwipeUniMag instance to interact with the Shuttle device.

There are several flags and methods available for the Shuttle. For an app that does not need much specific control of the swipe device and is mostly interested in the swipe event, the defaults can be kept and the device will power up and become ready for swipe when attached.

- *id delegate;*

  Gets or sets the delegate that will receive events. You should not set this directly. When the PGSwipeController's delegate is set, it will pass it through to this delegate.

- *PGSwipeUniMagMessageOptions *messageOptions;*

Contains a set of options for interactions with the user, e.g. whether to prompt before powering up the Shuttle and the text of error messages. See the PGSwipeUniMagMessageOptions section for specific settings.

- *BOOL activateReaderOnAttach;*

  If this is true, the SDK will attempt to power-up the reader when attachment is detected. There are 3 things to be aware of:

  1. If the user attaches headphones to the mobile device, it will be treated as a swipe reader and an attempt to power it up will be made.
  2. Before the attempt to activate the reader, if messageOptions.activateReaderWithoutPromptingUser is set to false (it is false by default), the user will receive a prompt asking to confirm activation. If they decline, no activation will be attempted.
  3. If you call powerDown to deactivate the device, leaving activateReaderOnAttach set to true will cause the device to immediately power back up.
- *BOOL automaticallySetVolumeToMaxOnActivate;*

  If this is set to true, the device's volume will be set to maximum immediately before any attempt to power on the reader. Since the reader requires full volume to activate, this defaults to true and should normally remain true.

- *BOOL alwaysAcceptSwipe;*

  The Shuttle does not accept swipes from the user unless a swipe has been requested. If alwaysAcceptSwipe is true, the SDK will immediately request a swipe and renew the request any time the old swipe request times out or ends. You will still receive periodic didBecomeUnreadyForSwipe: messages, but the reason will be SwipeReasonUnreadyForSwipeRefreshing to indicate that you should be receiving a didBecomeReadyForSwipe: message immediately after without any interaction.

  The mobile device's battery may deplete faster if the swipe reader is always awaiting a swipe. If battery life is a concern, consider setting this to false and using requestSwipe when a swipe is expected, or only setting alwaysAcceptSwipe to true when a swipe is expected.

  If alwaysAcceptSwipe is true, you should not use requestSwipe or cancelSwipeRequest. By default, alwaysAcceptSwipe is true.

- *-(void)powerUpDevice:(BOOL)powerUp;*

  Powers up the reader if powerUp is true or cancels a power up if powerUp is false. If activateReaderOnAttach is true, this is called automatically after connection to power up the device. If you wish to only power up the device after user interaction, you should wait until a deviceConnected: event is received, then call powerUpDevice:YES when they choose to power up. This should only be used if activateReaderOnAttach is false.

- *-(void)powerDown;*

  Powers down the reader. This may extend mobile device battery life. A deviceConnected event will be received after shut-down, which will trigger a power-up if activateReaderOnAttach is true, so be sure to set activateReaderOnAttach to false before powering down. It is not necessary to power down the reader before disconnecting it from the device.

- *-(void)requestSwipe;*

  Starts listening for swipe events. You will never need to call this if you set alwaysAcceptSwipe to true (it is true by default). After receipt of a didBecomeUnreadyForSwipe message, you may request a new swipe (unless the reason is SwipeUnreadyForSwipeReasonRefreshing). The request will timeout after 20 seconds, or the amount of time you set in setSwipeTimeoutDuration:.

- *-(void)cancelSwipeRequest;*

  Cancels a swipe request to stop listening for swipe events. You should not use this if you did not manually start the swipe request with a call to requestSwipe.

- *-(void)setSwipeTimeoutDuration:(int)seconds;*

  Sets the time between requestSwipe and when swipes will no longer be accepted. Default and maximum are 20 seconds. The minimum is 3 seconds. This still applies even if alwaysAcceptSwipe is true, but the swipe request will be automatically renewed in that case.

PGSwipeUniMagMessageOptions

This class contains a set of user-interaction options for the Shuttle device.

- *BOOL activateReaderWithoutPromptingUser;*

  If this is set to true (false by default), the reader is activated automatically immediately after you receive a deviceConnected: event. If this is false, you will need to call powerUpDevice: during or after the deviceConnected event to power the device or cancel powering up.

- *BOOL showInitializingReaderMessage;*

  If true, an "Initializing Card Reader..." alert is shown while the reader powers up and is dismissed once power-up completes.

- *NSString *cardReaderActivationPrompt;*

  Gets or sets the prompt that will be displayed to confirm that the user would like to power up the reader. If you change this prompt, you should also change

cardReaderActivationAtMaxVolumePrompt. Note: an activation prompt is only shown before activation if messageOptions.activateReaderWithoutPromptingUser is false. This message is meant to include a warning to indicate that the volume will be set to max. If automaticallySetVolumeToMaxOnActivate is false, cardReaderActivationAtMaxVolumePrompt is shown instead of this.

- *NSString *cardReaderActivationAtMaxVolumePrompt;*

  Gets or sets the prompt that will be displayed to confirm that the user would like to power up the reader. If you change this prompt, you should also change cardReaderActivationPrompt. Note: an activation prompt is only shown before activation if messageOptions.activateReaderWithoutPromptingUser is false. If automaticallySetVolumeToMaxOnActivate is true and the volume is not at maximum, cardReaderActivationPrompt is shown instead of this.

- *NSString *cardReaderTimeoutMessage;*

  Gets or sets the alert that is shown if the reader times out while attempting to power up. If you prefer to handle this differently, set this message to nil to prevent it from being shown, then handle the activationDidComplete:result: message with a result of SwipeActivationResultTimeout.

- *NSString *volumeTooLowMessage;*